پروژه درس معماری پیشرفته

مروری بر تحقیقات گذشته

استاد:

خانم دکتر جاسبی

دانشجو:

ندا باغبان کاشانی

(۳۹۹۱۲۳۴۱۰۵۷۰۲۴)

خردادماه ۱۴۰۰

عنوان مقاله:

Optimized data storage algorithm of IoT based on cloud computing in distributed system

الگوریتم ذخیره‌سازی داده بهینه شده IoT براساس محاسبات ابری در سیستم توزیع‌شده

(Wang and Zhang, 2020) – Computer Communications

تعریف مساله و هدف اصلی مقاله:

محاسبات ابری عمدتا مبتنی بر فن‌آوری اینترنت است و به کاربران خدمات، کاربردها و تعاملات ارائه می‌دهد. در میان آن‌ها، "ابر" استعاره‌ای برای اینترنت و شبکه است، که عمدتا یک بازنمایی انتزاعی از اینترنت و زیرساخت‌ها است. تعاریف زیادی برای محاسبات ابری وجود دارد و بسیاری از آن‌ها توسط موسسه فن‌آوری آمریکا پذیرفته شده‌اند. امروزه، محاسبات ابری یک مدل مبتنی بر پرداخت است که دسترسی راحت و در دسترس، به محض تقاضای شبکه را فراهم می‌کند و آن نیاز به تلاش کمتری برای تعامل با فروشنده در صورت نیاز به دسترسی به یک مجموعه قابل پیکربندی از منابع محاسباتی دارد. علاوه بر این، محاسبات ابری مزایای زیادی دارد، مانند مقیاس فوق بزرگ، مجازی‌سازی، قابلیت اطمینان بالا، تطبیق پذیری، مقیاس‌پذیری بالا، خدمات مورد نیاز، هزینه پایین و غیره. علاوه بر این، محاسبات ابری می‌تواند علاوه بر خدمات محاسباتی، خدمات ذخیره‌سازی را نیز ارائه دهد. بنابراین با توجه به مزایای متعدد آن، محاسبات ابری به طور گسترده‌ای در بسیاری از زمینه‌ها مورد استفاده قرار می‌گیرد. بسیاری از شرکت‌ها از محاسبات ابری برای ذخیره اطلاعات دسترسی به داده خود استفاده می‌کنند که دارای حریم خصوصی مشخصی است، اما هنوز هم خطرات بالقوه ای وجود دارد. بنابراین، بهینه‌سازی ذخیره‌سازی دسترسی داده‌ها در محاسبات ابری امروزه به یکی از موضوعات کلیدی تحقیقاتی در میان محققان تبدیل شده‌است.

به‌عنوان یکی از خدمات بسیار ارائه‌شده توسط محاسبات ابری، ذخیره‌سازی ابری است که به دلیل کارایی بالا، انعطاف‌پذیری و سیستم پرداخت به‌جای شما، به‌طور گسترده‌ای مورد استقبال قرار گرفته و به کاربران اجازه می‌دهد تا داده‌ها را ذخیره و بر روی پلتفرم به اشتراک بگذارند. با این حال، داده‌های ذخیره‌شده در بستر ابر در یک دامنه غیر قابل کنترل قرار دارند و مالک داده‌ها (Data Owner-DO) کنترل داده‌ها را از دست می‌دهد، که خطر زیادی برای امنیت به همراه دارد. هنگامی‌که کاربران می‌خواهند داده‌ها را از طریق پلتفرم ابری پس از اینکه داده‌های رمزگذاری شده در پلتفرم ابری آپلود می‌شوند، به دست آورند، آن‌ها قوانین دسترسی خودشان را دارند. اگر مکانیزم رمزگذاری کلید عمومی قدیمی در هنگام آپلود داده‌ها با عملیات رمزگذاری اتخاذ شود، مالک داده (DO) باید یک عملیات رمزگذاری داده را برای هر کاربر اجرا کند که نه تنها بار محاسباتی در سمت DO را افزایش می‌دهد، بلکه ارائه‌دهنده خدمات ابری (Cloud Service

Provider-CSP) باید این عملیات را هر بار پس از به اشتراک‌گذاری داده‌های ذخیره شده تکرار کند. برای حل مشکلات بالا، یک طرح رمزگذاری مبتنی بر اسناد (Attributed-Base Encryption, ABE) که از انتخاب برخی صفات و ویژگی‌های هویت کاربران یا همه صفات برای رمزگذاری و رمزگشایی داده‌ها استفاده می‌کند، توسط Sahai و Waters پیشنهاد شد. ABE به DO ها اجازه می‌دهد تا داده‌ها را بدون استفاده از کلیدهای عمومی دیگران به اشتراک بگذارد، که مقدار محاسبات را کاهش می‌دهد، اما نمی‌تواند از حملات تبانی در میان چند کاربر جلوگیری کند. علاوه بر این، براساس طرح ABE، طرح رمزگذاری مبتنی بر ویژگی Key-Policy یا سیاست کلیدی (KP-ABE) که در آن مالک داده (DO) ساختار دسترسی را در کلید کاربر ایجاد می‌کند، و مجموعه‌ای از صفات و ویژگی‌ها برای انجام عملیات رمزگذاری بر روی داده‌ها مورد استفاده قرار گرفت، توسط Goyal و همکاران ارائه شد که در آن کنترل دسترسی بسیار دقیق و انعطاف‌پذیری در مدیریت حقوق کاربر را می‌توان به دست آورد، اما این راه‌حل، به‌روز رسانی تمام کلیدهای کاربران را به‌هنگام ابطال کلید نیاز دارد. علاوه بر آن، براساس متن‌رمزی و ساده‌سازی فرآیند ابطال کلید، Betten-court و همکاران ویژگی رمزگذاری مبتنی بر سیاست متن‌رمزی را پیشنهاد دادند. (CP-ABE)، که سربارهی محاسباتی آن در مقایسه با طرح KP-ABE بسیار بیشتر بود

سرویس به اشتراک‌گذاری داده ابری طراحی شده برای افزایش حفاظت از حریم خصوصی توسط Sabrina و همکاران پیشنهاد شد که ساختار کنترل دسترسی را به درخت کنترل دسترسی باینری تغییر داد. با این حال، به دلیل محدودیت‌های ساختاری، انعطاف‌پذیری راه‌حل محدود بود و کارایی ذخیره‌سازی به طور قابل‌توجهی بهبود نمی‌یابد.

براساس فن‌آوری استخراج داده و الگوریتم ژنتیک، Karimi و همکاران ترکیبی از خدمات آگاه از QoS (QoS-aware) را در محاسبات ابری پیشنهاد کردند که الزامات آن باید به صورت پویا پیاده‌سازی شوند، و نیاز به توازن بین بهترین وضعیت عملکرد و سرعت اجرای سبد خدمت دارد. به منظور دستیابی به این هدف، ترکیبی از روش‌ها در مطالعات قبلی برای به حداکثر رساندن و بهینه‌سازی نتایج در کوتاه‌ترین زمان ممکن مورد استفاده قرار گرفته است. با این حال، با افزایش تعداد خدمات و گسترش فضای جستجو برای مشکلات، کارایی کافی برای روش‌های سنتی در ترکیب خدمات مورد نیاز در زمان معقول وجود نداشت. بنابراین، الگوریتم ژنتیک برای بهینه‌سازی سطح خدمات در سطح جهانی مورد استفاده قرار می‌گیرد. علاوه بر این، خوشه‌بندی خدمات برای کاهش فضای جستجو در مساله مورد استفاده قرار می‌گیرد، و با توجه به تاریخچه قوانین وابستگی برای خدمات ترکیبی برای بهبود بهره‌وری ترکیب خدمات مورد استفاده قرار می‌گیرد. Shila و همکاران یک ذخیره‌سازی امن موبایل را به‌نام سیستم ذخیره‌سازی محاسبات ابری پیشنهاد کردند (Lunqiang, 2018). اگرچه این تحقیقات پیشین محاسبات ابری و بهینه‌سازی داده‌های مطمئنی را به اشتراک گذاشته‌اند، مدل‌های محاسبات ابری عمدتا خدمات محاسبات ابری را با مجموعه‌ای از ماشین‌های اختصاصی و گران‌قیمت ارائه می‌دهند که می‌تواند منجر به سرمایه‌گذاری‌های عظیم در هزینه‌های سرمایه و هزینه‌های جاری شود. محاسبات ابری یک پارادایم انقلابی با استفاده موثر از منابع و قابلیت مدیریت پیشرفته است که خدماتی را از ذخیره‌سازی داده‌ها و پردازش تا فرآیند منابع محاسباتی نرم‌افزار در شبکه فراهم می‌کند.

امروزه، مدل محاسبات ابری متداول شامل مجموعه‌ای از ماشین‌های اختصاصی و گران‌قیمت است که خدمات محاسبات ابری را فراهم می‌کند و منجر به سرمایه‌گذاری‌های عظیم در هزینه‌های اولیه و هزینه‌های جاری می‌شود. راه‌حل مقرون‌به‌صرفه، استفاده از قابلیت‌های ابرهای تک‌کاره متشکل از منابع محلی توزیع‌شده و دینامیک توسعه‌نیافته است، و این مقاله یک راه‌حل سیستم توزیع‌شده را اتخاذ می‌کند. همچنین می‌تواند به ابرهای ثابت و متحرک تقسیم شود. ابرهای ثابت از منابع محاسباتی استفاده می‌کنند که توسط ماشین‌های با اهداف عمومی کم‌تر مورد استفاده قرار می‌گیرند و ابرهای متحرک از منابع محاسباتی بلااستفاده دستگاه‌های موبایل استفاده می‌کنند. با این حال، ماهیت دینامیکی و توزیع‌شده ابره‌ای تک کاره، چالشی را برای مدیریت سیستم ایجاد می‌کند (Yu, 2019).

تا آنجا که به تحقیقات موجود مربوط می‌شود، الگوریتم‌های ذخیره‌سازی دسترسی داده‌ها در محاسبات ابری دارای نقص کم در پردازش موثر داده‌ها و تحمل خطای کم هستند که نمی‌تواند نیازهای جامعه امروز را برای ذخیره‌سازی دسترسی داده‌ها برآورده کند. بنابراین، HDFS که به یک سیستم فایل توزیع‌شده اشاره دارد که تحمل خطا بالایی دارد و می‌تواند به ماشین‌های ارزان متصل شود، برای طراحی الگوریتم بهینه‌سازی ذخیره‌سازی دسترسی به داده‌ها در محاسبات ابری معرفی شده‌است. در همین حال، HDFS همچنین می‌تواند دسترسی داده‌های با توان عملیاتی بالا را فراهم کند که برای برنامه‌های ذخیره‌سازی اطلاعات دسترسی به داده‌ها ایده‌آل است. کاربرد HDFS تا حد زیادی می‌تواند کارایی پردازش داده و نرخ تحمل خطای الگوریتم بهینه‌سازی ذخیره دسترسی داده IoT را بهبود بخشد، که فن‌آوری‌های موثرتری را برای ذخیره اطلاعات دسترسی به داده IoT فراهم می‌کند.

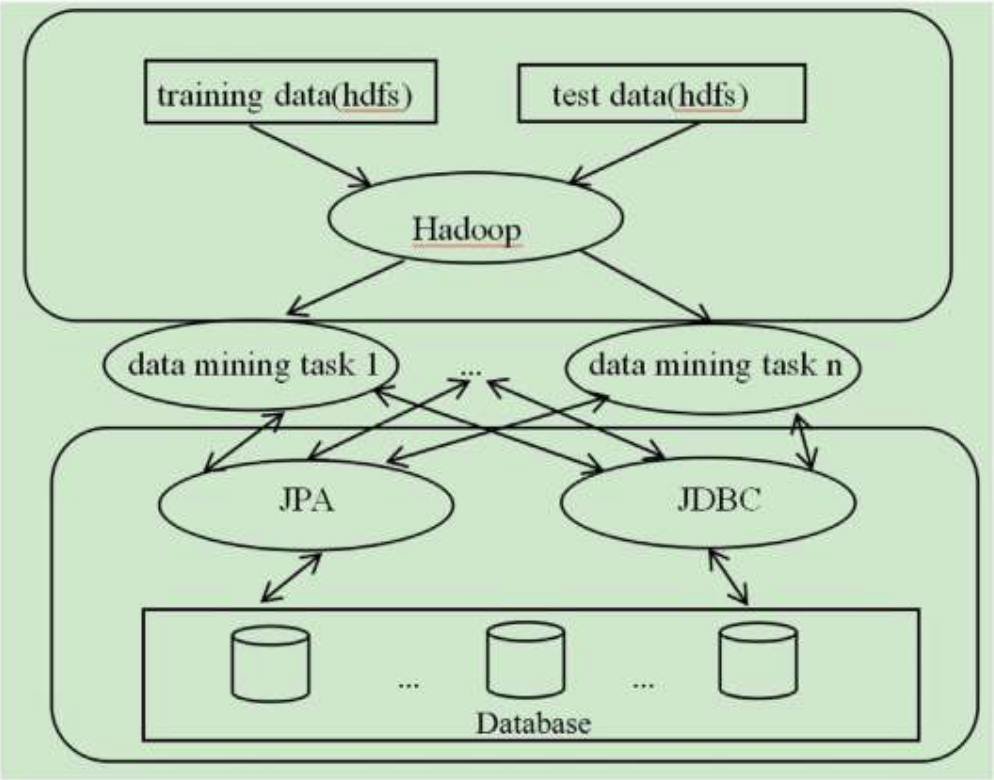
توضیح راه‌حل پیشنهادی مقاله برای حل مساله:

۱- ساختار استخراج داده در سیستم ذخیره‌سازی ابر بزرگ

محاسبات ابری یک سیستم کاربردی Web 2.0 سازنده و یک پلت فرم اطلاعاتی مبتنی بر محاسبات ابری را پیاده‌سازی می‌کند. به منظور دستیابی به دسترسی به منابع و استخراج داده برای سیستم ذخیره‌سازی ابر بزرگ، ایجاد یک مدل کلی در ابتدا ضروری است. سپس از طریق ساخت دسترسی به منابع در سیستم ذخیره‌سازی ابر در مقیاس بزرگ، تبادل اطلاعات و یکپارچه‌سازی منابع داده‌های چند منبعی محقق شده و خدمات کسب وکار چند بعدی و کنترل چند عملکردی نیز ارائه می‌شوند. در نهایت، سیستم ذخیره‌سازی ابر تحت سیستم دیجیتال سرویس اطلاعات توسط پرس و جوی داده‌های چند کاناله و داده‌های شبکه از طریق مکانیزم مدیریت حافظه نظارت می‌شود که می‌تواند کارایی و امنیت دسترسی و زمان‌بندی داده‌های شبکه را بهبود بخشد. سرویس اطلاعات توسط پرس و جوی داده‌های چند کاناله و داده‌های شبکه از طریق مکانیزم مدیریت حافظه که می‌تواند کارایی و امنیت دسترسی و زمان‌بندی داده‌های شبکه را بهبود بخشد، نظارت می‌شود. علاوه بر این، به منظور ایجاد تعادل دینامیکی سیستم دسترسی QoS و خدمت در اطلاعات دیجیتال

لایه MAC، مدل کلی چارچوب داده کاوی در پلتفرم هادوپ در این مقاله ساخته شده‌است، که در شکل ۱ نشان‌داده شده‌است.

در شکل ۱، اتصال پایگاه داده جاوا (JDBC) نشان‌دهنده یک رابط عملیاتی خواندن و نوشتن تصادفی تحت سیستم ذخیره‌سازی ابر بزرگ است، که ابزارهای استانداردسازی را برای خواندن و نوشتن برنامه‌ها در سیستم ذخیره‌سازی ابر در زمانی که برنامه‌ریزی منابع انجام می‌شود فراهم می‌کند، و قابلیت پردازش موازی دارد. یکپارچه‌سازی منابع برای سیستم‌های ذخیره‌سازی ابر بزرگ با محاسبات ابری به عنوان هسته. با این حال، عملیات پرس و جوی داده‌های بزرگ مکرر در سیستم ذخیره‌سازی ابری منجر به افزایش فشار بار بر روی داده‌های ذخیره‌شده در موتور تجزیه می‌شود و طراحی برنامه‌ریزی در دسترسی به منابع، مورد نیاز خواهد بود. برای استخراج داده وظیفه (task) ۱ به وظیفه n، زمانی که y(n) توالی زمانی گاوسی است که هر عنصر به طور مستقل از عدد شبه تصادفی گاوسی در پاسخ پرس و جوی سیستم ذخیره‌سازی ابر بزرگ تولید می‌شود، $A=\{A_1, A_2,\ldots, A_m\}$ ، شاخص طبقه‌بندی داده‌ها است و بیش‌ترین مقدار اعتماد (trust value) برابر با ۱ است، هر چه بازه زمانی طولانی‌تر باشد، تاثیر ارزش اعتماد بر وضعیت فعلی در طراحی داده کاوی و برنامه‌ریزی منابع در ترکیب با داده‌ها در نمودار چارچوب کم‌تر خواهد بود و تابع کاهش زمان برابر با $T_{sim} \epsilon$ $\{0, 1\}$ است. علاوه بر این، بردار معکوس لبه به دست آمده برای نشان دادن رتبه داده های اصلی x(n) استفاده می شود، و عبارت سازماندهی مجدد داده‌های چندمنظوره ای است که به‌طور موازی توسط سرویس اطلاعات دیجیتال تحت سیستم ذخیره سازی ابر پرس و جو از طریق انتقال فوریه با دامنه قابل تنظیم حاصل می‌شود. سپس ، سیستم ذخیره سازی ابر با کمک Hadoop ساخته می‌شود تا جریان داده‌های عظیم بعدی را استخراج کند. سرانجام، از اجرای کد توزیع شده و روش نمایه‌سازی داده‌های درخت KD برای توزیع کد کار در گره نظارت بر چند مسیر استفاده می‌شود.



شکل ۱- ساختار استخراج داده در پلتفرم Hadoop

در این مقاله، بر اساس تئوری تفکیک تابع همبستگی خودکار، تابع جایگزینی حافظه نهان در این مقاله ساخته شده تا تعادل دینامیکی را در درخت KD ایجاد کند، و تابع جایگزینی حافظه نهان به‌دست آمده به شرح زیر است.

$$MTTA = \sum_{i,j,l=1,1,1}^{M,n,N} d_{ijl} \cdot Q\left(d_{ij}\right)^{-1} \cdot T\left(s_l\right) / (N-1) \qquad (1)$$

که در آن d_{ijl} پارامتر حالت برنامه‌ریزی منابع در سیستم ذخیره‌سازی ابر با فاصله‌بندی برابر، $Q(d_{ij})$ بردار شاخه برنامه کنترل خطی بلوک داده و $T(S_l)$ تابع زمانی حرکت گره در سیستم ذخیره‌سازی ابر است. به عبارت دیگر، استخراج داده در سیستم‌های ذخیره‌سازی ابر بزرگ از طریق مدل ساختار طراحی فوق‌الذکر محقق می‌شود، که یک پایه و اساس دقیق برای دسترسی به زمان‌بندی منابع فراهم می‌کند.

۲- طراحی الگوریتم بهینه‌سازی برای ذخیره‌سازی دسترسی به داده‌های IoT

اطلاعات دسترسی به داده برای هر دامنه یا دستگاه حیاتی است. مشتریان و دستگاه‌ها را می توان براساس اطلاعات دسترسی داده‌ها برای اطمینان از رضایت مشتری و امنیت دستگاه تحلیل کرد. بنابراین، اطلاعات دسترسی به داده‌ها باید ذخیره شوند، و سیستم ذخیره‌سازی دسترسی به داده‌های IoT ویژگی‌های زیر را دارد:

(۱) اجرای عملی (Practical performance): با توجه به اطلاعات مختلف داده‌ها، درگاه داده‌ها باید فراهم شود، و سیستم نظارت کامل برای نظارت زمان واقعی استفاده می‌شود، که ارتباط نزدیک با دنیای خارج را حفظ می‌کند و تجربه خوب استفاده از کاربران را تضمین می‌کند.

(۲) هزینه پایین (Low cost): تجهیزات توسعه سیستم ذخیره‌سازی دسترسی به داده‌های IoT ساده است، و هزینه‌های نگهداری و اقتصادی با توسعه مداوم فن‌آوری در حال کاهش و کاهش است.

(۳) مقیاس پذیری (Scalability): تعداد سرورها برای سیستم‌های ذخیره‌سازی دسترسی به داده‌های IoT می‌تواند به صدها مورد برسد، و هر چه تعداد بیشتر باشد، سیستم ذخیره‌سازی مقیاس‌پذیرتر خواهد بود.

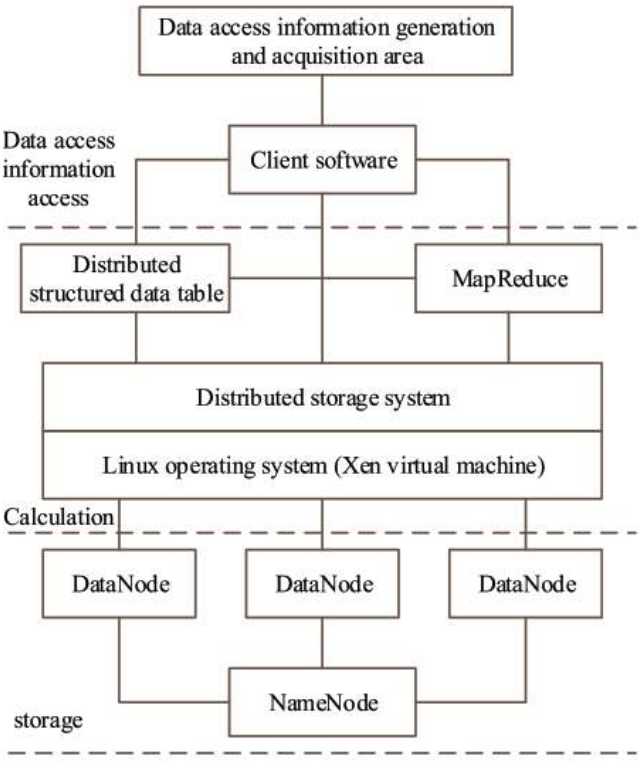
همه موارد ذکر شده در بالا الزامات بالاتری را بر روی الگوریتم ذخیره‌سازی دسترسی به داده‌های IoT قرار داده‌اند. با این حال، الگوریتم های ذخیره دسترسی داده‌های موجود نمی‌توانند این الزامات را برآورده کنند، بنابراین آن‌ها باید بر این اساس بهینه‌سازی شوند. خلاصه فرآیند خاص آن به‌شرح زیر است:

۲-۱- بهینه‌سازی معماری ذخیره‌سازی دسترسی به داده

به منظور بهبود کارایی پردازش داده الگوریتم ذخیره‌سازی، HDFS ابتدا با توجه به ویژگی‌های اطلاعات دسترسی به داده IoT، مزایای رایانش ابری و الزامات ذخیره‌سازی دسترسی به داده، برای بهینه‌سازی معماری ذخیره‌سازی دسترسی به داده معرفی می‌شود. معماری ذخیره‌سازی دسترسی داده‌ها همانطور که در شکل ۲ نشان‌داده شده، بهینه‌سازی شده‌است. همچنانچه در شکل ۲ نشان داده شده پس از بهینه‌سازی معماری ذخیره‌سازی دسترسی به داده‌ها، این معماری به سه بخش اصلی تقسیم می‌شود که عبارتند از: دسترسی به اطلاعات، محاسبه و ذخیره‌سازی داده‌ها.

نرم‌افزار مشتری برای دسترسی به اطلاعات دسترسی به داده‌ها می‌تواند اطلاعات دسترسی را ایجاد کند و حوزه‌های عملکردی را به‌دست آورد. علاوه بر این، هسته محاسباتی، یک سیستم ذخیره‌سازی توزیع‌شده با سیستم‌عامل لینوکس است، که ماشین مجازی Xen که بخش کلیدی آن بخش ذخیره‌سازی است. خوشه HDFS که توسط NameNode مدیریت می‌شود، متعاقبا اطلاعات دسترسی به داده‌ها را ذخیره می‌کند و می‌تواند اطلاعات دسترسی به داده‌های عظیم را ذخیره و مدیریت کند.

اطلاعات دسترسی به داده‌های IoT را می‌توان به فایل HDFS انتقال داد یا توسط دسترسی به اطلاعات دسترسی به داده‌های IoT و بخش محاسبه با توجه به الزامات مختلف در پایگاه داده ذخیره کرد، که هر دوی آن‌ها می‌توانند ذخیره‌سازی مداوم اطلاعات دسترسی به داده‌ها را تحقق بخشند. در همین حال، HDFS که همزمان به آن اشاره شد می‌تواند تا حد زیادی تاخیر دسترسی به داده‌ها را کاهش داده و تجربه کاربر را افزایش دهد.



شکل ۲- نمودار بهینه‌سازی معماری ذخیره‌سازی دسترسی به داده

۲-۲- بهینه‌سازی استراتژی در توزیع ذخیره‌سازی دسترسی به داده‌ها

براساس معماری ذخیره‌سازی دسترسی به داده بهینه بالا، الگوریتم hash برای بهینه‌سازی مجدد استراتژی توزیع دسترسی به داده تطبیق داده می‌شود. در فرآیند ذخیره‌سازی اطلاعات دسترسی به داده‌ها، HDFS برای انجام ذخیره‌سازی توزیعی برای اطلاعات دسترسی به داده‌ها معرفی می‌شود. با این حال، یک مساله کلیدی این است که چگونه اطلاعات دسترسی به داده‌ها را بر روی گره‌های IoT توزیع کنیم، یعنی استراتژی توزیع ذخیره‌سازی دسترسی به داده‌ها که تاثیر مستقیم و بزرگی بر روی کارایی پردازش داده‌ها دارد.

به منظور بهبود کارایی پردازش داده، الگوریتم hash با عملکرد بالا و هزینه نگهداری پایین بر روی ساختار داده برای بهینه‌سازی استراتژی توزیع دسترسی به داده استفاده می‌شود، که می‌تواند تا حد زیادی شکست گره‌های IoT و انتقال اطلاعات دسترسی به داده ناشی از افزایش گره‌ها را کاهش دهد.

۱-۲-۲- عوامل موثر بر توزیع ذخیره‌سازی دسترسی به داده‌ها

پس از دسترسی به اطلاعات دسترسی داده، می توان آن را به چندین بلوک داده تقسیم کرد و براساس قوانین خاص بر روی گره‌های IoT توزیع کرد. از طریق تحقیق مشخص شد که عوامل اصلی موثر بر توزیع ذخیره‌سازی دسترسی داده‌ها به طور عمده به سه دسته تقسیم می‌شوند:

(۱) تعادل بار (Load balancing): در فرآیند توزیع ذخیره‌سازی، اطلاعات دسترسی به داده‌ها باید به طور مساوی بر روی گره‌ها توزیع شوند تا تعادل بار گره‌ها حفظ شود، به طوری که اطلاعات دسترسی به داده‌ها بتوانند به صورت موازی پردازش شوند تا کارایی پردازش داده‌ها بهبود یابد. اگر توزیع ذخیره‌سازی متعادل نباشد، "انحراف داده‌ها" در اطلاعات دسترسی به داده‌ها وجود خواهد داشت. بدتر اینکه، انجام پردازش موازی برای گره‌ها غیرممکن خواهد بود، که تاثیر بیشتری بر ذخیره اطلاعات دسترسی به داده‌ها خواهد داشت.

(۲) خطای گره (Node failure): برای خوشه‌های HDFS، این یک مشکل مکرر است که گره‌های آن خارج از نظم هستند. بنابراین، الگوریتم بهینه‌سازی در ذخیره‌سازی دسترسی به داده‌ها باید تحمل خطای بسیار بیشتری داشته باشد، به طوری که اطلاعات دسترسی به داده‌ها را بتوان به شیوه‌ای متعادل ذخیره کرد.

(۳) اجرای عملیات ذخیره‌سازی (Storage operation performance): تاثیر ذخیره‌سازی دسترسی داده‌ها به طور مستقیم تحت‌تاثیر اجرا و عملکرد عملیات است. به طور کلی، تنها عملیات انتقال شبکه در نظر گرفته می‌شود. اگر عملیات در انتقال شبکه ساده باشد، ارتباطات میان گره‌ها در فرآیند اجرایی الگوریتم ذخیره‌سازی دسترسی داده می‌تواند تا حد زیادی کاهش یابد و کارایی فشرده‌سازی اطلاعات دسترسی داده می‌تواند به طور موثر بهبود یابد.

با توجه به تجزیه و تحلیل بالا و همبستگی بین اطلاعات دسترسی به داده‌ها، الگوریتم هش برای نگاشت و ذخیره داده‌های مرتبط در همان گره اعمال می‌شود و اطلاعات دسترسی به داده‌های مرتبط جمع‌آوری می‌شود، به طوری که پرس و جو و تجزیه و تحلیل بر روی اطلاعات دسترسی به داده‌های بعدی تسهیل می‌شود.

۲-۲-۲- پیکربندی مقدار Hash برای دسترسی داده‌ها به موقعیت ذخیره اطلاعات

نمودار شماتیک الگوریتم Hash در شکل ۳ نشان‌داده شده‌است. همچنانکه در شکل ۳ ملاحظه می‌شود الگوریتم هش به چهار حوزه تقسیم می‌شود: مدول A، مدول B، مدول C و مدول D که متناظر با

DataNode1، DataNode3، DataNode5 و DataNode7 است. حلقه الگوریتم Hash با محاسبه مقدار Hash تکمیل می‌شود.

از آنجایی‌که حلقه الگوریتم Hash ، استراتژی توزیع در ذخیره‌سازی دسترسی داده بهینه‌سازی شده‌است. مراحل خاص به شرح زیر هستند:

اول، ارتباط اطلاعات دسترسی به داده و تعداد نسخه‌های اضافی باید تنظیم شوند و فرمول محاسبه درجه همبستگی بر روی اطلاعات دسترسی به داده عبارت است از:

$$\delta = \frac{\sum_{i=1}^{n} x_i * \alpha}{n^2 + 1} \qquad (2)$$

که δ نشان‌دهنده درجه همبستگی بین اطلاعات دسترسی به داده‌ها است، α تعداد نسخه‌های اضافی را نشان می‌دهد که با توجه به ادبیات فنی موضوع و تحقیقات گذشته بر روی ۳ تنظیم شده است و n تعداد کل اطلاعات دسترسی به داده‌ها می‌باشد.

دوم، مقدار گره Hash در خوشه HDFS بر این اساس محاسبه و در حلقه Hash پیکربندی می‌شود. فرمول محاسبه روی مقدار Hash گره‌ها به صورت زیر بیان می‌شود:

$$\omega = \int_{t=1}^{m} \sqrt[2]{p} \otimes \beta \qquad (3)$$

که در آن ω مقدار Hash گره‌ها، p گره، m تعداد کل گره‌ها را نشان می‌دهد و β پارامتر محاسبه مقدار Hash گره‌ها را نشان می‌دهد.

سپس با توجه به همبستگی اطلاعات دسترسی به داده‌ها، مقدار Hash داده‌ها به طور متناظر محاسبه می‌شود و فرمول محاسبه به صورت زیر بیان می‌شود.

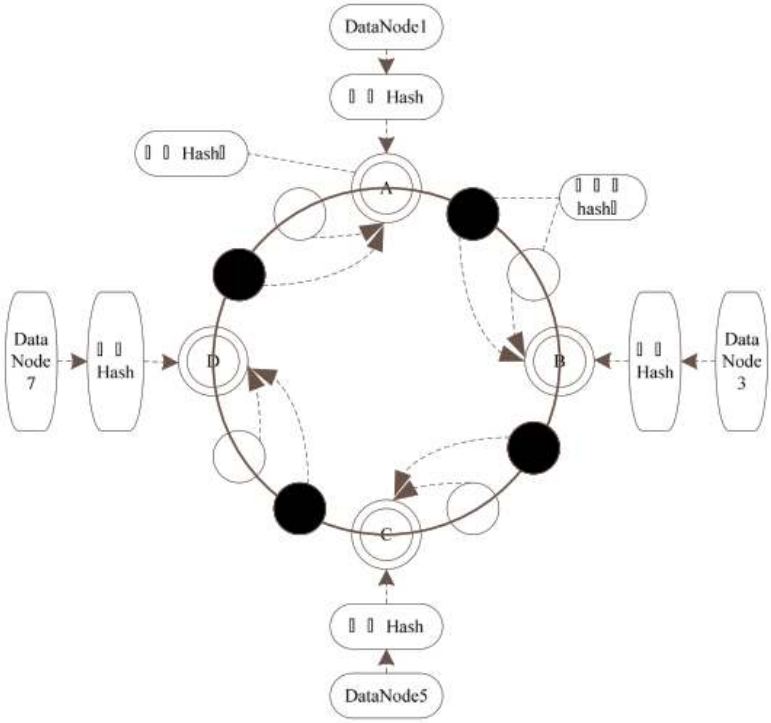
$$\mu = \frac{\int_{i=1}^{n} \sqrt[2]{x} \otimes \delta}{n^2} \qquad (4)$$

که در آن μ نشان‌دهنده مقدار Hash دسترسی به داده‌ها است.

در نهایت، مکان ذخیره اطلاعات دسترسی به داده‌ها با توجه به مقدار Hash به‌دست‌آمده از گره‌ها و داده‌ها پیکربندی می‌شود و نتیجه پیکربندی به صورت زیر بدست می‌آید:

$$f(x) = \prod_{i=1,t=1} {}^{\omega}/_{\mu} * \chi \qquad (5)$$

که در آن، χ پارامتر پیکربندی را نشان می‌دهد.



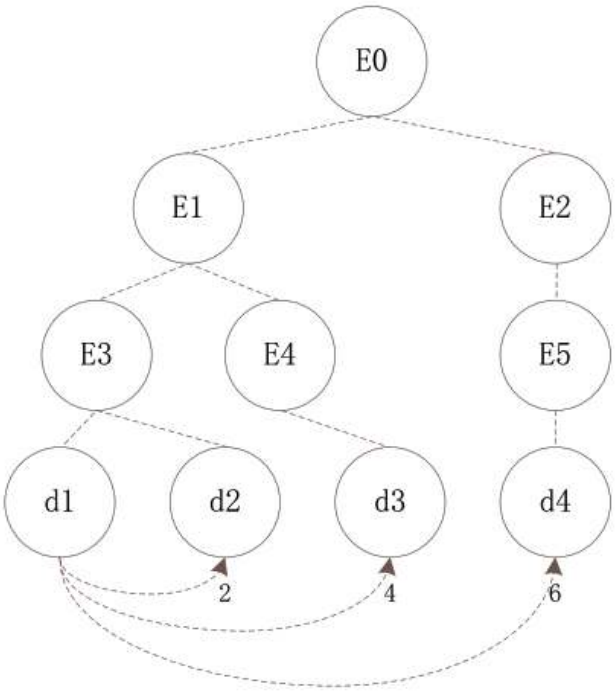
شکل ۳- نمودار شماتیک الگوریتم Hash

۳-۲- بهینه‌سازی توپولوژی شبکه IoT

از آنجایی که این شبکه توسط محاسبات ابری در فرآیند ذخیره‌سازی دسترسی به داده‌ها ذخیره می‌شود، توپولوژی شبکه تاثیر زیادی بر روی اثر ذخیره‌سازی خواهد داشت. یک توپولوژی شبکه خوب می‌تواند تا حد زیادی کارایی ذخیره دسترسی داده را بهبود بخشد و ذخیره دسترسی داده را سرعت بخشد. توپولوژی‌های شبکه الگوریتم ذخیره‌سازی دسترسی به داده‌های IoT موجود مشکلات عمده‌ای دارند. بنابراین، توپولوژی شبکه برنامه‌ریز برای بهینه‌سازی استفاده می‌شود. فرآیند خاص بهینه‌سازی به شرح زیر است:

گره‌ها در توپولوژی شبکه عمدتا در یک درخت مرتب شده‌اند و هر گره (مانند d1، d2، ...) به گره سوئیچینگ (مانند E0، E1، E2، ...)کامپیوتر متصل شده است. این نمودار بهینه‌سازی توپولوژی شبکه در شکل ۴ نشان

داده شده است. همانطوریکه در شکل ۴ مشاهده می‌شود، فاصله بین هر گره و سوئیچ، کوتاه‌ترین فاصله پس از بهینه‌سازی توپولوژی شبکه IoT است، که می‌تواند تا حد زیادی سرعت دسترسی به اطلاعات و ذخیره‌سازی داده‌ها را بهبود بخشد، و عملکرد ذخیره‌سازی اطلاعات دسترسی به داده‌ها را بهبود بخشد. علاوه بر این، بهینه‌سازی در توپولوژی شبکه را می‌توان با مزیت بهتری بدست آورد.



شکل ۴- نمودار بهینه‌سازی توپولوژی شبکه IoT

۴-۲- بهینه‌سازی اندازه بلوک داده

ذخیره دسترسی داده‌ها با استفاده از HDFS، در واقع، داده‌های دسترسی را به بلوک‌های داده برای ذخیره‌سازی توزیع‌شده تقسیم می‌کند. بنابراین، مشاهده می‌شود که اندازه بلوک داده‌ها نیز به طور مستقیم بر ذخیره‌سازی دسترسی به داده‌ها تاثیر می‌گذارد. به منظور بزرگ‌تر و سریع‌تر کردن ذخیره‌سازی دسترسی داده‌ها، الگوریتم اثر برای بهینه‌سازی اندازه بلوک داده استفاده می‌شود. فرمول بهینه‌سازی برای اندازه بلوک داده عبارت است از:

$$effect(\gamma) = f(x) \wp \frac{trans\_time}{trans\_time - seek\_time} \qquad (6)$$

که در آن transe_time دسترسی به اطلاعات دسترسی به داده‌ها و seek_time زمان پیکربندی محل ذخیره اطلاعات دسترسی به داده‌ها است.

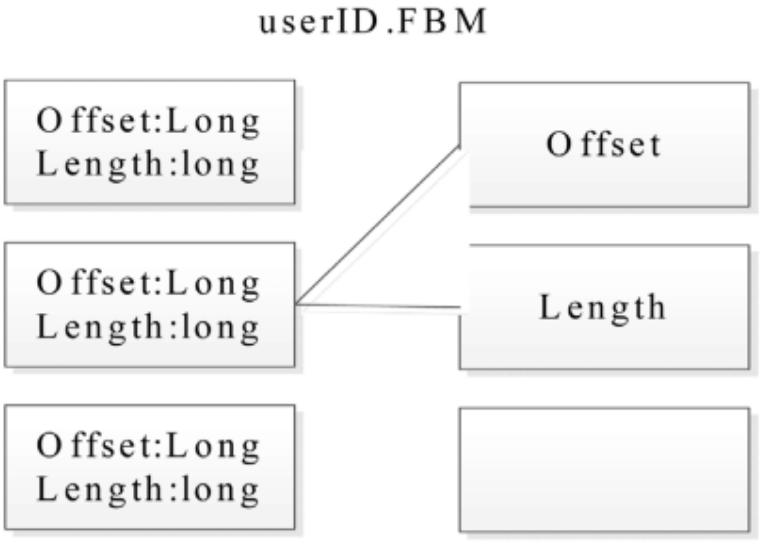
با توجه به فرمول بالا، اندازه بهینه بلوک داده به دست می‌آید و اطلاعات دسترسی داده به طور متناظر براساس آن ذخیره می‌شوند.

از طریق فرآیند بالا، بهینه‌سازی ذخیره‌سازی دسترسی داده‌ها در رایانش ابری محقق می‌شود که می‌تواند تا حد زیادی کارایی پردازش داده‌ها و تحمل خطا در ذخیره‌سازی دسترسی داده‌ها را بهبود بخشد و پشتیبانی فنی پیشرفته‌تری را برای ذخیره‌سازی و مدیریت دسترسی داده‌ها فراهم کند.

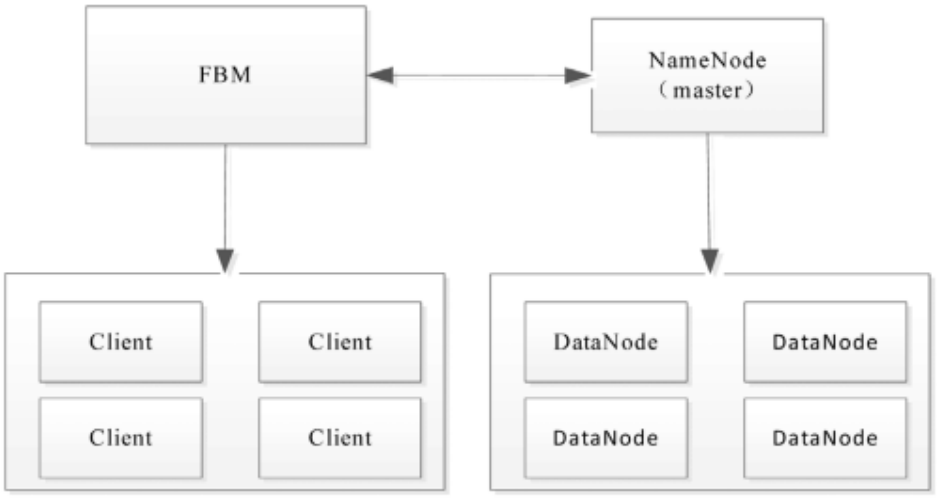
۳- روش‌های بهینه‌سازی ذخیره‌سازی فایل

۳-۱- طرح بهینه‌سازی ذخیره‌سازی فایل

معماری اصلی HDFS و حالت مدیریت بلوک برای معرفی متا سرور اطلاعات File-Block-Mapping (FBM) برای ذخیره‌سازی فضای کاربر IoT نگهداری می‌شوند. علاوه بر این، FBM اندازه فایل و مقدار offset فایل کاربر ذخیره‌شده در HDFS را ثبت می‌کند، و هر کاربر روی HDFS فایل کاربری خود را دارد و تمام فایل‌های خود را ذخیره می‌کند. علاوه بر آن، هر نام فایل مربوط به ID کاربر است، که رابطه نگاشت آن در شکل ۵ و بهینه‌سازی فایل در شکل ۶ نشان‌داده شده‌است. طرح فایل بهینه عمدتا به مشتریان و گره‌های داده به‌ترتیب از طریق FBM و گره اصلی دسترسی دارد. تاثیر بهبود یافته براساس طرح مدیریت موازی می‌تواند زمان پاسخ سیستم را کوتاه کند. و اطلاعات فایل متا داده یک سیستم ذخیره، بیشتر اطلاعات متا فایل سیستم را اشغال می‌کند، اما آن اطلاعات مقدار کمی از حافظه ذخیره را اشغال می‌کند. بنابراین، این نوع سیستم ذخیره‌سازی فایل به عنوان هدف بهینه‌سازی استفاده می‌شود.



شکل ۵- ارتباط نگاشت (Mapping) بین فایل‌های FBM و فایل‌های کاربر

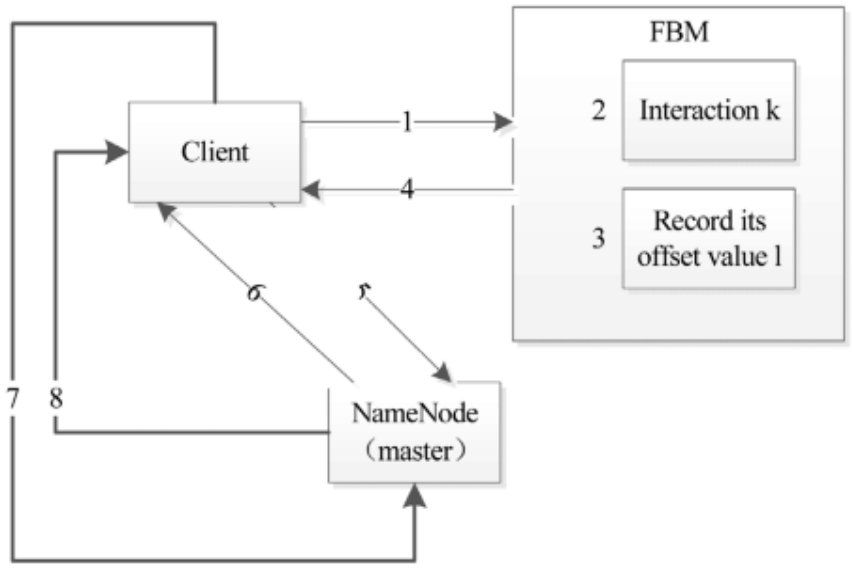


شکل ۶- ارتباط نگاشت (Mapping) بین فایل‌های FBM و فایل‌های کاربر

۲-۳- فرآیند خواندن و نوشتن برنامه بهینه‌سازی فایل

(۱) فرآیند نوشتن فایل در شکل ۷ نشان‌داده شده است.

مشتری یک درخواست ارسال می‌کند، در (۱) فایل درخواست j نوشته می‌شود، که با NameNode تعامل ندارد، اما در (۲) با FBM تعامل دارد. سپس در (۳) مقدار افست (۱) ثبت می‌شود و در (۴) FBM به عنوان رکورد موفقیت m مشخص می‌شود. در همین حال، در (۵) مشتری جریان نوشتن فایل‌های کاربر را به NameNode ارسال می‌کند، که در (۶) این جریان را به مشتری باز می‌گرداند. در این زمان، در (۷) مشتری با NameNode تعامل می‌کند تا عملیات نوشتن داده واقعی p را اجرا کند، و در (۸) فایل جدید برای پایان q می‌نویسد.

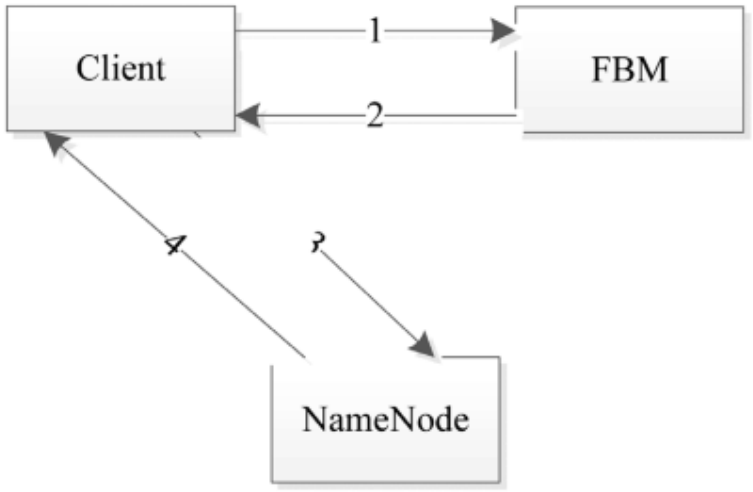


شکل ۷- نمودار فرآیند نوشتن فایل

(۲) فرآیند خواندن فایل کاربر در شکل ۸ نشان‌داده شده است.

مشتری با FBM و گره نام در یک روش دو طرفه تعامل می‌کند و سه طرف می‌توانند به طور کامل با یکدیگر ارتباط برقرار کنند. ابتدا، مشتری درخواست خواندن نام فایل را می‌کند، سپس به طور مستقیم با FBM تعامل می‌کند تا اطلاعات متا داده را به دست آورد. ابتدا، در (۱) مشتری درخواست خواندن نام فایل را می‌کند، سپس در (۲) به طور مستقیم با FBM تعامل می‌کند تا اطلاعات متا داده را به دست آورد. علاوه بر این، مشتری خواستار خواندن جریان ورودی فایل کاربر در (۳) با توجه به (۲) است، و فایل بازگشتی از NameNode خوانده می‌شود، که به مشتری در (۴) ارائه می‌شود.

(۳) کاربران IoT فایل‌ها را بدون حذف مستقیم داده‌های فایل اصلی حذف می‌کنند، اما اطلاعات متا داده‌های خود را در FBM ذخیره می‌کنند، که برای حذف پرچم تنظیم شده است.



شکل ۸- خواندن فایل‌ها

۳-۳- اجرای ذخیره‌سازی بهینه فایل

به منظور بهینه‌سازی ذخیره‌سازی فایل‌های کوچک، کلاس SmallFileStatus به اطلاعات فایل کوچک متا-داده براساس HDFS اصلی معرفی می‌شود، که مشخصه‌های آفست و اندازه فایل‌های کوچک را در فایل‌های کاربر براساس FileStatus اضافه می‌کند. علاوه بر این، کلاس اصلی طراحی بهینه‌سازی ذخیره‌سازی فایل کوچک، کلاس FBM است، ویژگی‌های اصلی آن نقشه فایل است که متعلق به نوع HashMap است. علاوه بر این، جفت مقدار-کلید نام فایلی است که کلید آن فایل کوچکی است، و مقدار آن اطلاعات متا داده است.

در این مقاله HDFS بر روی یک ماشین فیزیکی ساخته شده، و یک سناریو آزمایش شبیه‌سازی در OPNET Modeler انجام شده که به ۳ گره سرور IoT مجهز شده و برای انجام آزمایش‌ها به به تجهیزات ارتباطی متصل شده است.

نتایج آزمایش‌های انجام شده در مقاله نشان می‌دهد که داده‌های اینترنت اشیا بهینه شده توسط الگوریتم پیشنهادی در این مقاله از نظر سرعت انتقال، اشغال منابع سیستم و زمان پاسخ نسبت به حالت اصلی برتری دارد. علاوه بر این، حداکثر بازده پردازش داده‌های اینترنت اشیا از انتقال بهینه شده می‌تواند به ۹۹٪ و حداکثر میزان تحمل خطا از الگوریتم بهینه سازی شده می تواند به ۹۶,۱۲٪ برسد.

نقاط قوت و ضعف مقاله

۱- مشکلات ناشی از راه‌حل بهینه‌سازی فایل و مزیت راه‌حل ارائه شده در مقاله

طبق توضیحات بخش قبل روش به‌کار برده شده در مقاله، کاربران اینترنت اشیا (IoT) بارها و بارها عملیات اضافه و حذف کردن را انجام می‌دهند و تقسیم‌بندی زیادی را ایجاد خواهند کرد. بنابراین، یک آستانه، که برابر با نسبت قسمت بندی فایل به تمام فایل‌ها است، تنظیم می‌شود. با توجه به اینکه از حد مجاز تجاوز می‌شود، سیستم بخشی از آن را مرتب خواهد کرد. به عبارت دیگر، یک فایل کاربر جدید برای آن کاربر ایجاد خواهد شد که فایل اصلی کاربر در آن کپی خواهد شد و فایل شاخص FBM مربوطه نیز برای فایل جدید ایجاد خواهد شد. علاوه بر این، فایل جدید، فایل کاربر اصلی نامیده می‌شود، و فایل شاخص جدید متناظر نیز به جای فایل اصلی به فایل شاخص اصلی تغییر نام می‌یابد.

هنگامی که سیستم روشن می‌شود، FBM اطلاعات فایل متنی هر کاربر و ورودی‌های شاخص فایل کوچک را بارگذاری می‌کند. سپس، سیستم اطلاعات متا داده کاربر را در حافظه بارگذاری می‌کند تا مشتری بتواند اطلاعات فایل را در مسیر فایل قرار دهد. عیب آن این است که اطلاعات فراداده بیش از حد در حافظه بارگذاری می‌شوند، که ممکن است منجر به سربار حافظه FBM بیش از حد بالا شود. برای حل مشکل ذخیره‌سازی مداوم FBM، پایگاه‌داده (database) رابطه‌ای برای ذخیره اطلاعات مرتبط متا داده در فضای پایگاه داده اتخاذ می‌شود، که مزایای به‌کارگیری آسان و کاهش مصرف حافظه FBM را دارد، اما پایگاه‌های داده رابطه‌ای نمی‌توانند در یک خوشه مستقر شوند. علاوه بر این، یک نقطه محدودیت ممکن است پایگاه‌داده را تا حدی مستعد تنگناهای عملکرد کند. علاوه بر آن، پایگاه‌داده غیر رابطه‌ای NoSQL، مانند پایگاه‌داده MongoDB می‌تواند برای ذخیره فایل و اطلاعات پوشه مورد استفاده قرار گیرد. این پایگاه‌داده را می‌توان در یک خوشه توزیع‌شده گسترش داد، که مشکلات خاصی در استقرار دارد، اما پایگاه‌داده MongoDB می‌تواند به طور موثرتری مشکل محدودیت تک‌نقطه‌ای را حل کند.

هنگامی که کاربران جدید اینترنت اشیا ثبت‌نام می‌کنند، فایل‌های کاربری خودشان را خواهند داشت. فایل‌های کاربر به آرامی در یک سطح خاص جمع می‌شوند و تبدیل به فایل‌های بزرگ می‌شوند. بنابراین، در یک خوشه HDFS، زمانی که گره NameNode تنها دارای اطلاعات متا داده یک کاربر (userID.file) است، مصرف حافظه NameNode به‌میزان زیادی کاهش خواهد یافت.

هنگامی که کاربر یک عملیات را حذف می‌کند، یک پرچم حذف ۱ در userID.FBM تنظیم می‌شود، و userID.file نگه داشته می‌شود. به عبارت دیگر، زمانی که کاربر می‌خواهد به فایل IoT دسترسی داشته باشد، علامت تعیین موقعیت کاربر userID.FBM برای نشان دادن این که فایل وجود ندارد، استفاده خواهد شد. این فایل هنوز هم در HDFS وجود دارد که نمی توان به وسیله شاخص به آن دست یافت، که یک محیط قطعه قطعه و تقسیم شده است. علاوه بر این، یک فایل جدید در انتهای فایل کاربر اضافه می‌شود. این فرآیند به کار خود ادامه می‌دهد و تکه‌تکه شدن فایل‌های کاربر افزایش می‌یابد که منجر به کاهش کارایی

ذخیره‌سازی کل خوشه HDFS می‌شود. بنابراین، روش‌های مدیریت قسمت‌بندی مربوطه برای اجرای مدیریت قسمت‌بندی در فایل‌های کاربر اتخاذ می‌شوند.

آستانه فوق روی ۲۰ ٪ تنظیم می‌شود و یک محاسبه قطعه برای هر حذف انجام می‌شود، در حالی که در مقایسه با اندازه کل فایل، تا زمانی که مجموع اندازه فایل در اطلاعات متا داده با پرچم حذف ۱ بیشتر از ۲۰ ٪ باشد، برنامه مدیریت دیسک، برای نوشتن اطلاعات متا داده مربوط به پرچم حذف بیت به جز ۱ در یک فایل موقت، اجرا خواهد شد. در همین حال، ذخیره موجود فایل‌های کاربر بر روی فایل‌های موقت نوشته می‌شوند که به فایل اصلی تغییر نام داده و جایگزین آن می‌شوند.

جمع‌بندی و پیشنهادات برای کارهای آتی

اینترنت اشیا (IoT) موجود از الگوریتم‌های ذخیره‌سازی دسترسی به داده محاسبات ابری استفاده می‌کند، در واقع یک الگوریتم Hash (Hash Algorithm) است که الگوریتم درهم سازی دارای نقص‌های بهره‌وری پردازش داده کم و نرخ تحمل خطای کم است. بنابراین، HDFS برای بهینه‌سازی الگوریتم‌های ذخیره‌سازی دسترسی به داده محاسبه ابری معرفی می‌شود. HDFS ابتدا برای بهینه‌سازی معماری ذخیره‌سازی دسترسی به داده‌ها با توجه به مشکلات معماری ذخیره‌سازی دسترسی به داده‌ها در اینترنت اشیا استفاده می‌شود، که در آن عوامل توزیع ذخیره‌سازی دسترسی به داده‌ها در IoT به طور کامل در نظر گرفته می‌شوند، و مقادیر درهم‌سازی برای بهینه‌سازی پیکربندی مکان‌های ذخیره‌سازی دسترسی به داده‌ها مورد استفاده قرار می‌گیرد، به طوری که استراتژی توزیع دسترسی به داده‌ها می‌تواند بهینه شود. سپس توپولوژی IoT بهینه می‌شود و اندازه بلوک داده نیز با الگوریتم اثر بهینه می‌شود. در نهایت، طراحی ذخیره‌سازی فایل، بهینه شده است. از طریق آزمایش‌های شبیه‌سازی، ثابت شده است که روش ذخیره ابر بهینه‌شده دارای مزایای عملکرد واضحی در سرعت خواندن و نوشتن فایل و همچنین استفاده از حافظه است. در مقایسه با الگوریتم سنتی Hash، الگوریتم بهینه‌سازی پیشنهادی در این مقاله تا حد زیادی قابلیت بارگذاری و دانلود فایل، کارایی پردازش داده و نرخ تحمل خطا را بهبود می‌بخشد، که به طور کامل نشان می‌دهد که الگوریتم بهینه‌سازی ذخیره دسترسی به داده محاسبه ابری پیشنهادی، برتری بیشتری دارد.

در این مقاله به‌منظور به‌دست آوردن داده‌ها و نتایج تجربی دقیق‌تر، تحقیقات بیشتر در زمینه بهینه‌سازی الگوریتم بهینه ذخیره‌سازی اطلاعات IoT در رایانش و محاسبات ابری پیشنهاد شده است.

شبیه سازی

تست عملکرد الگوریتم بهینه‌سازی دسترسی به داده‌های ذخیره ابری IoT:

فرآیند طراحی الگوریتم بهینه‌سازی ذخیره‌سازی ابر ذکر شده در قبل برای دسترسی به داده‌های IoT را تحقق می‌بخشد، اما برای تعیین اینکه آیا می‌تواند مشکلات الگوریتم های موجود را حل کند یا خیر، نیاز به

تایید بیشتر دارد. بنابراین، در این مقاله یک آزمایش شبیه‌سازی به منظور مقایسه عملکرد الگوریتم بهینه‌سازی ذخیره‌سازی دسترسی داده‌ها در رایانش و محاسبات ابری طراحی شده‌است که عمدتا نشان‌دهنده عملکرد الگوریتم توسط کارایی پردازش داده‌ها و تحمل خطا می‌باشد.

۱- پیکربندی محیط

آزمایش برای ذخیره‌سازی بهینه فایل تنها NameNode را بهبود می‌بخشد، که DataNode ها را شامل نمی‌شود. بنابراین، HDFS بر روی یک ماشین فیزیکی ساخته می‌شود، و یک سناریوی آزمایش شبیه‌سازی در مدلساز OPNET Modeler ساخته می‌شود، که به ۳ گره سرور IoT مجهز است و برای انجام آزمایش‌ها به تجهیزات ارتباطی متصل می‌شود. پیکربندی سیستم در جدول ۱ نشان‌داده شده است.

جدول ۱- پیکربندی سیستم آزمایشی

	Configuration	Type
IoT server	CPU	Intel Xeon W-3175X 28-core 3.1G
	RAM	32GDDR4*4
IoT node	CPU	Intel i7-9700K Core Eight Core 3.6G
	RAM	16GDDR4*2
	hard disk	Seagate Galaxy Exos 7E88 TB 256 MB 7200 RPM

در طول آزمایش مقایسه شبیه‌سازی، به منظور اطمینان از دقت نتایج تجربی، پارامترهای خارجی در طول آزمایش یکسان نگه داشته می‌شوند. فرآیند تحلیل نتایج تجربی خاص در زیر نشان‌داده شده است.

۲- تحلیل نتیجه

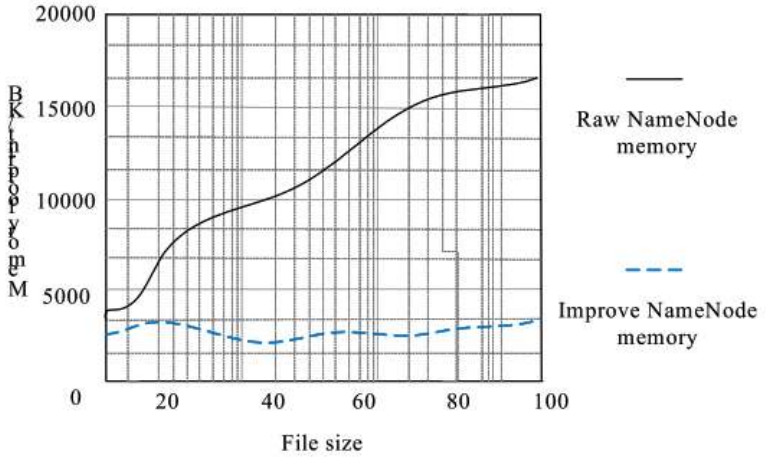
به منظور ارزیابی عملکرد روش افزودن فایل‌های بهینه‌سازی FBM به IoT، ۶ مجموعه آزمایش مقایسه‌ای بر روی HDFS بهبود یافته و HDFS اصلی انجام می‌شود. پارامترهای مورد آزمایش در جدول ۲ نشان‌داده شده‌اند.

جدول ۲- پارامترهای به‌کار برده شده در آزمایش

User scale	Number of files created	File size (w)
1000	100	10
2000		20
3000		40
4000		60
5000		80
6000		100

۱-۲- مقایسه حافظه فرآیند NameNode

اندازه حافظه فرآیند NameNode در طول آزمایش ثبت می‌شود، و نتایج آزمایش در شکل ۹ نشان داده شده است، که در آن محور افقی اندازه فایل و محور قائم مختصات اندازه حافظه NameNode را نشان می‌دهد که واحد آن KB است.

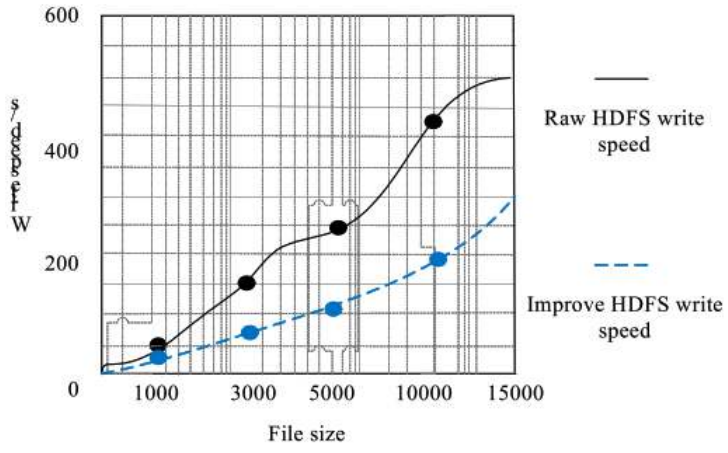


شکل ۹- مقایسه حافظه NameNode قبل و بعد از بهینه‌سازی فایل

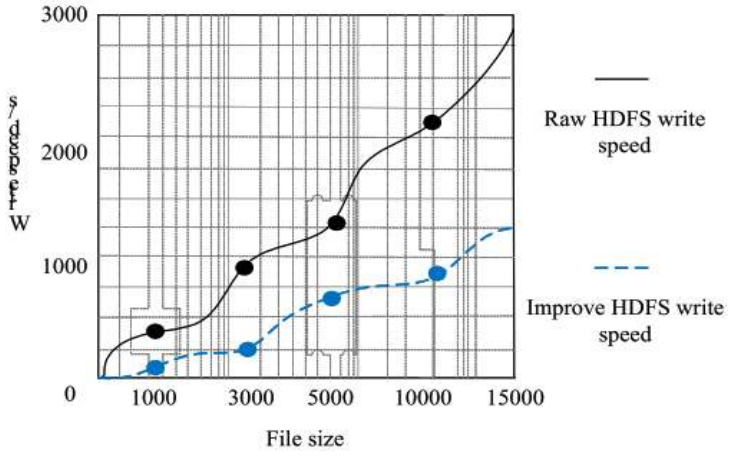
همانطوریکه در شکل ۹ مشاهده می‌شود خود فرآیند NameNode یک مصرف حافظه خاص دارد. در طول ذخیره‌سازی فایل، زمانی که اندازه فایل یکسان است، حافظه NameNode بزرگ‌تر از حافظه NameNode سیستم بهبود یافته است. علاوه بر این، با افزایش عدد اندازه فایل، مصرف حافظه NameNode بسیار سریع است، اما سیستم بهبود یافته مصرف حافظه NameNode نسبتا آهسته است و هیچ روند افزایشی قابل‌توجهی وجود ندارد. آزمایش‌ها نشان می‌دهند که از نظر مصرف حافظه، به‌ویژه هنگامی که اندازه فایل بیش از ۴۰ کیلوبایت است، اثر ردپای حافظه بهتر است، عملکرد طرح بهینه‌سازی ارائه‌شده در این مقاله بهتر از طرح اصلی است و فضای نام خوشه HDFS نیز گسترش می‌یابد.

۲-۲- مقایسه کارایی آپلود فایل

مقیاس خواندن و نوشتن فایل به ترتیب بر روی اندازه‌های ۱۰۰۰، ۳۰۰۰، ۵۰۰۰، ۱۰۰۰۰ و ۱۵۰۰۰ کیلوبایت تنظیم شده است و زمان صرف شده برای خواندن و نوشتن فایل مورد آزمایش قرار گرفته است. دو راه‌حل قبل و بعد از بهبود از نظر ایجاد فایل‌های بدون نوشتن داده مقایسه شده‌اند و نتایج مقایسه در شکل ۱۰ نشان‌داده شده است. مطابق شکل ۱۰ تحت اطلاعات فراداده‌ای (متا داده) مشابه، تفاوت بین دو طرح در دوره قبل مشخص نیست. همانطور که اطلاعات متا داده افزایش می‌یابد، سرعت نوشتن دو طرح برای ایجاد فایل‌ها به آرامی کاهش می‌یابد. با این حال، هنگامی‌که HDFS بهبود یافته طرح بهینه‌سازی HDFS ، برای نوشتن اطلاعات متاداده NameNode استفاده می‌شود، همانطور که اندازه فایل افزایش می‌یابد، سرعت نوشتن هم افزایش می‌یابد، اما سرعت این افزایش آهسته است، که حدود ۵۰ ٪ کم‌تر از روش اولیه است. مزیت سرعت ایجاد فایل‌ها هنگام نوشتن اطلاعات متا داده NameNode بسیار واضح است. آزمایش بالا در مورد سرعت ایجاد فایل‌ها است. به منظور تست سرعت نوشتن بلوک‌های داده واقعی، مقایسه‌های زیر انجام می‌شود. همچنان‌که از شکل ۱۱ ملاحظه می‌شود با افزایش اطلاعات واقعی، سرعت دو طرح در نوشتن داده‌های واقعی به آرامی کاهش می‌یابد. هنگامی که اندازه واقعی داده‌ها یکسان باشد، راه‌حل بهینه‌سازی با اضافه کردن FBM زمان بیشتری را برای نوشتن داده‌های واقعی نسبت به راه‌حل اصلی صرف می‌کند، زیرا HDFS عملیات فایل‌های ضمیمه را پشتیبانی نمی‌کند.



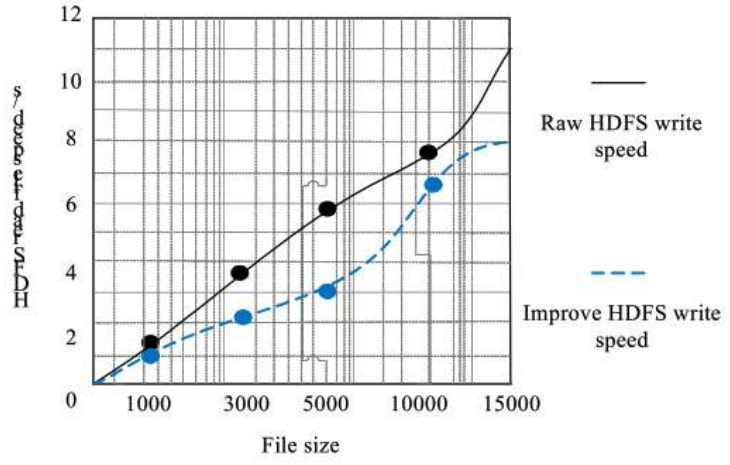
شکل ۱۰- مقایسه زمان صرف شده برای ساختن یک فایل (بدون نوشتن داده)



شکل ۱۱- مقایسه زمان صرف شده برای ساختن یک فایل (با نوشتن داده)

۳-۲- مقایسه کارایی دانلود فایل

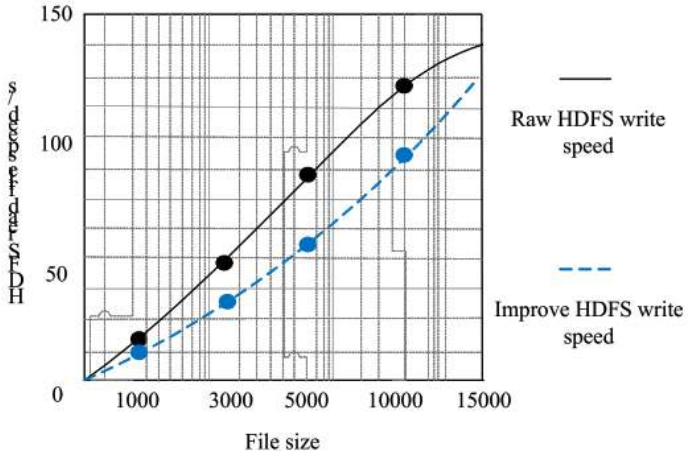
سرعت خواندن اطلاعات متا داده دو طرح با یکدیگر مقایسه شده است و اندازه فایل مشابه بخش قبل ۲-۲ است. سرعت در حالت بدون دانلود داده‌ها در شکل ۱۲ و سرعت با دانلود داده‌ها در شکل ۱۳ نشان‌داده شده است. همچنانچه از شکل‌های ۱۲ و ۱۳ قابل مشاهده است، طرح بهبود یافته که در آن FBM معرفی می‌شود، می‌تواند سرعت خواندن فایل‌های HDFS را تا حدی بهبود بخشد. به طور خلاصه، روش بهبود یافته پیشنهادی در این مقاله، یعنی روشی که در آن ساختار داده FBM معرفی می‌شود، راه‌حل ادغام فایل‌ها در فایل‌های بزرگ امکان‌پذیر و بهتر از روش اصلی HDFS است.



شکل ۱۲- مقایسه زمان صرف شده برای خواندن فایل‌ها (بدون دانلود داده)

۴-۲- تجزیه و تحلیل مقایسه‌ای بر روی کارایی پردازش داده‌ها

کارایی پردازش داده به طور مستقیم نشان‌دهنده کارایی الگوریتم است. به طور کلی، هرچه بازده پردازش داده بالاتر باشد، عملکرد الگوریتم بهتر خواهد بود. مقایسه کارایی پردازش داده به‌دست‌آمده از طریق آزمایش‌ها در جدول ۳ نشان‌داده شده‌است. در جدول ۳ نشان‌داده شده‌است که وقتی‌که تکرار آزمایش کوچک است، تنها ۲۰ بار، کارایی پردازش داده الگوریتم موجود ۵۶٪ است، و بازده پردازش داده‌های الگوریتم بهینه به ٪ ۸۹ می‌رسد. علاوه بر این، با افزایش تکرار آزمایش، اگر تعداد آزمایش‌ها به ۲۰۰ برسد، کارایی پردازش داده الگوریتم موجود ۶۱ ٪ است و کارایی پردازش داده الگوریتم بهینه می‌تواند به ۹۳ ٪ برسد. بدیهی است که کارایی پردازش داده الگوریتم بهینه بسیار بیشتر از الگوریتم موجود با تکرارهای آزمایش مختلف است. علاوه بر این، الگوریتم بهینه می‌تواند به حداکثر کارایی پردازش که ۹۹ ٪ پس از ۱۸۰ بار آزمایش است، برسد.



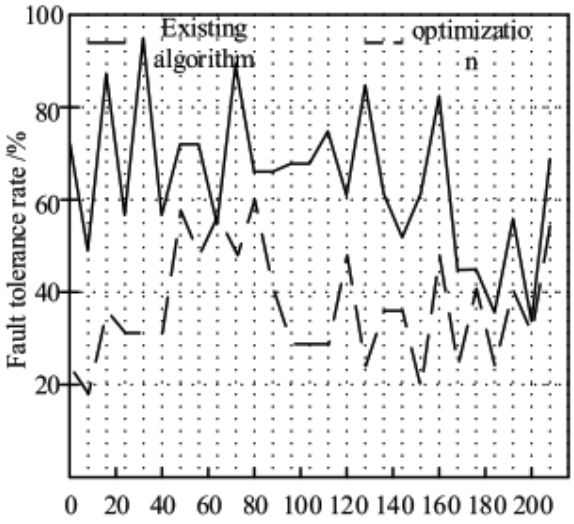
شکل ۱۳- مقایسه زمان صرف شده برای خواندن فایل‌ها (با دانلود داده)

جدول ۳- مقایسه کارایی پردازش داده‌ها

Experiment times	Existing algorithm	Optimization algorithm
20	56%	89%
40	64%	88%
60	50%	79%
80	49%	76%
100	67%	85%
120	73%	81%
140	62%	91%
130	53%	90%
180	69%	99%
200	61%	93%

۵-۲- تجزیه و تحلیل مقایسه‌ای در تحمل خطا

از آنجایی که خطاهای گره به طور مکرر در HDFS رخ می‌دهند، تحمل خطای بیشتر می‌تواند عملکرد ذخیره‌سازی الگوریتم را بهبود بخشد. مقایسه تلورانس خطا به‌دست‌آمده از طریق آزمایش‌ها در شکل ۱۴ نشان‌داده شده است. همچنانکه در شکل ۱۴ مشاهده می‌شود تحمل خطا در الگوریتم موجود از ۱۹ ٪ تا ۶۰ ٪ متغیر است و الگوریتم بهینه دارای حداقل تحمل خطا ۳۴ ٪ و حداکثر ۹۶,۱۲ ٪ می‌باشد. بنابراین، تحمل خطا در الگوریتم بهینه بسیار بیشتر از الگوریتم های موجود است.



شکل ۱۴- مقایسه تحمل پذیری خطا

با توجه به نتایج تجربی ذکر شده در بالا، الگوریتم بهینه‌سازی ذخیره‌سازی دسترسی داده‌های IoT در محاسبات ابری پیشنهادی در این مقاله تا حد زیادی بازدهی پردازش داده‌ها و تحمل خطا را بهبود می‌بخشد، که به طور کامل نشان می‌دهد که الگوریتم بهینه‌سازی ذخیره‌سازی دسترسی داده‌های IoT در محاسبات ابری پیشنهادی در این مقاله عملکرد بهتری دارد.